Security Review Report
NM-0244 Pyth Data Association

NETHERMIND
SECURITY

(July 12, 2024)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind for the Pyth Network Cairo contracts. `Pyth Network` is an oracle protocol that connects the owners of market data to applications on multiple blockchains. Pyth's market data is contributed by over 100 first-party publishers, including some of the biggest exchanges and market-making firms in the world. Over 350 protocols on 55+ blockchains trust Pyth to secure their applications.

This review focuses on the price feeds created by the `Pyth` Network in the Starknet network. Because Wormhole does not officially support Starknet, a wormhole contract to receive the necessary messages was also developed and reviewed.

The audited code comprises 2450 lines of code. The `Pyth` team has provided comprehensive documentation explaining the behavior of protocol and the reviewed smart contracts. Aside from the provided documentation, the `Pyth` and `Nethermind Security` teams have communicated to clarify any remaining questions about the expected behavior of the protocol.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract. Along with this document, we report 1 points of attention, classified as Low The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 describes the system overview. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.

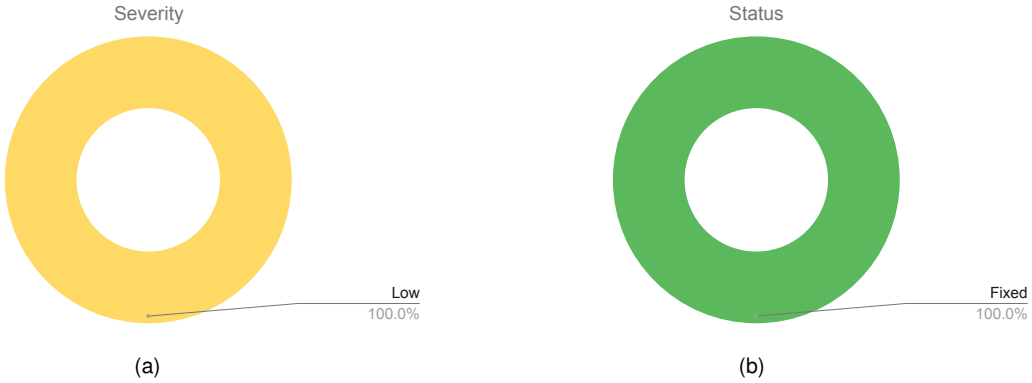

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (1), **Undetermined** (0), **Informational** (0), **Best Practices** (0). **Distribution of status: Fixed** (1), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0),

Summary of the Audit

Audit Type	Security Review
Initial Report	July 12, 2024
Final Report	July 12, 2024
Repository	pyth-network/pyth-crosschain
Commit (Audit)	`11506d931ced152c6e02d4b35271b10528d0d8ad`
Commit (Final)	80194334a0be12b32926211416f6318381dfb473
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/wormhole.cairo	211	38	18.0%	22	271
2	src/util.cairo	169	18	10.7%	20	207
3	src/pyth.cairo	665	60	9.0%	68	793
4	src/byte_buffer.cairo	127	25	19.7%	17	169
5	src/hash.cairo	102	16	15.7%	7	125
6	src/lib.cairo	15	0	0.0%	1	16
7	src/reader.cairo	166	26	15.7%	13	205
8	src/merkle_tree.cairo	56	2	3.6%	10	68
9	src/util/exp10_.cairo	83	0	0.0%	0	83
10	src/pyth/fake_upgrades.cairo	50	15	30.0%	14	79
11	src/pyth/errors.cairo	105	7	6.7%	14	126
12	src/pyth/interface.cairo	103	12	11.7%	12	127
13	src/pyth/price_update.cairo	203	12	5.9%	27	242
14	src/pyth/governance.cairo	165	20	12.1%	16	201
15	src/wormhole/errors.cairo	76	9	11.8%	7	92
16	src/wormhole/parse_vm.cairo	57	4	7.0%	7	68
17	src/wormhole/interface.cairo	39	50	128.2%	13	102
18	src/wormhole/governance.cairo	58	16	27.6%	8	82
	Total	**2450**	**330**	**13.5%**	**276**	**3056**

3 Summary of Issues

	Finding	Severity	Update
1	Wormhole contract may start with stale `guardian_set_index`	Low	Fixed

4 System Overview

Pyth is a protocol that allows market participants to publish pricing information on-chain for others to use. The protocol is an interaction between three parties:

- Data providers submit pricing information to Pyth's oracle program. Pyth has multiple data providers for every price feed to improve the accuracy and robustness of the system.
- Pyth's on-chain oracle program on Pythnet combines providers' submitted data to produce a single aggregate price and confidence interval.
- Applications read the price information produced by the oracle program. More specifically, Pyth allows users to "pull" prices onto the blockchain when needed. Those prices become publicly available for everyone on that chain.

The reviewed contracts contain two main contracts that allow pricing information to be delivered and consumed in the Starknet network, the Pyth and the Wormhole contract.

4.1 Pyth

The `Pyth` contract is the contract used for consuming the price information. The trusted data sources emit messages through the Wormhole protocol indicating the new valid price information.

The message emitted by the trusted sources contains the root from a Merkle tree containing all the valid price information updates. Any user can use this message to submit one of the price feed updates validated by the Merkle root.

The following functions are some of the main endpoints accessible when interacting with this contract:

- The ***get_price_unsafe(...)***, ***get_ema_price_unsafe(...)***, and ***query_price_feed_unsafe(...)*** functions allow users to get the latest updated information for a specific price feed. These functions will revert if the price feed does not exist.
- The ***get_price_no_older_than(...)***, ***get_ema_price_no_older_than(...)***, and ***query_price_feed_no_older_than(...)*** functions allow users to get the latest updated information for a specific price feed and it will revert if the price feed has not been updated for more than a specified time provided by the user.
- The ***update_price_feeds(...)*** function allows users to submit price feed updates that are validated by trusted sources. Price feeds are only updated if the new information was published at a timestamp greater than the current one.
- The ***update_price_feeds_if_necessary(...)*** function allows the user to submit price feeds that are validated by trusted sources. Price feeds will only be updated if the current price feeds are not updated when compared with the specified timestamps. Price feeds are only updated if the new information was published at a timestamp greater than the current one.
- The ***parse_price_feed_updates(...)*** function allows users to submit price feeds that are validated by trusted sources and get the price feed information from the used message for a set of specified price ids. Users can also specify that they only want to get price feed information published in a specific time frame. Like the previous functions, price feeds are only updated if the new information was published at a timestamp greater than the current one.
- The ***parse_unique_price_feed_updates(...)*** function works similarly to ***parse_price_feed_updates(...)*** with the difference that it will only return price feeds information that are the first update for the price feed in a specific timeframe.
- The ***execute_governance_instruction(...)*** is used to submit messages emitted by the Pyth governance. These messages execute the following administrative actions:
  - The ***SetFee*** and ***SetFeeInToken*** actions change the fee paid for executing updates.
  - The ***SetDataSources*** action changes the set of trusted data sources. These are the sources allowed to emit messages validating price feed updates.
  - The ***SetWormholeAddress*** action is used to change the address of the Wormhole contract. The Wormhole contract is the mechanism used by Pyth Network for cross-chain communication.
  - The ***AuthorizeGovernanceDataSourceTransfer*** is used for the process of changing the Pyth Network governance.
  - The ***UpgradeContract*** permits the upgrade of the Pyth contract, effectively changing its implementation.

The contract also exposes other functions that allow users to retrieve more information about the system.

4.2 Wormhole

`Wormhole` allows `Pyth` to scale easily to multiple blockchains while keeping data sourcing and on-chain aggregation on the `Pythnet` appchain. As a generic message-passing protocol, `Wormhole` can bring `Pyth` price outputs as transactions to EVM chains, Aptos, Cosmos Hub, Starknet, and more.

`Pyth` price updates are created on `Pythnet` and streamed off-chain via the Wormhole Network, a cross-chain messaging protocol. These updates are signed such that the `Pyth` on-chain program can verify their authenticity. This allows updating the on-chain price to be a permissionless operation: anyone can submit a valid `Wormhole` message to the `Pyth` contract to update the price.

Until the review date, `Wormhole` has not deployed a contract to verify messages in `Starknet`. Because of this, the Pyth team has created their own `Wormhole` contract to verify these messages.

The following functions are some of the main endpoints accessible when interacting with this contract:

- The ***parse_and_verify_vm(...)*** function is the main entry point of this contract. This function is used to verify that a specific message was indeed sent through the `Wormhole` protocol. The function will also parse and return the submitted data in a structured way.
- The ***get_guardian_set(...)*** and ***get_current_guardian_set_index(...)*** functions allow to get information about the current set of guardians.
- The ***submit_new_guardian_set(...)*** is used to submit a new guardian set. This action can only be triggered by the `Wormhole` governance entity. When a new guardian set is submitted, the previous guardian set is still valid for 1 day after this message is submitted.

The contract also exposes other functions that allow users to fetch more information about the contract.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Low] Wormhole contract may start with stale `guardian_set_index`

File(s): src/wormhole.cairo

Description: The `wormhole` contract stores the `current_guardian_set_index`, which indicates the guardian set index currently accepted by the contract. When the contract is deployed, the `current_guardian_set_index` is initialized to `zero`, as shown in the following code snippet:

```
#[constructor]
fn constructor(
    ref self: ContractState,
    initial_guardians: Array<EthAddress>,
    chain_id: u16,
    governance_chain_id: u16,
    governance_contract: u256,
) {
    self.chain_id.write(chain_id);
    self.governance_chain_id.write(governance_chain_id);
    self.governance_contract.write(governance_contract);
    let set_index = 0;
    self.store_guardian_set(set_index, @initial_guardians);
}
```

However, the currently active guardian set in the `Wormhole` protocol may differ from the one initialized at deployment. Consequently, all previous messages updating the guardian set index from zero to the current active one need to be submitted.

When an active guardian set is replaced, the previous set remains valid for one day. Because of this, after initialization, all the previous guardian sets would be valid for one day. This would allow a possible malicious previous set of guardians to submit false messages to the `Pyth` contract.

Recommendation(s): Consider passing the index of the current guardian set as an argument to the contract's constructor to ensure accuracy from deployment.

Status: Fixed.

Update from the client: Fixed in commit 80194334a0be12b32926211416f6318381dfb473.

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about the Pyth Network documentation

The documentation for Pyth Network is public available on their website. Beyond this, the Pyth team has provided extra documentation related to how messages are formatted and sent. The team has been available to answer any doubts or questions raised by the Nethermind team.

8 Test Suite Evaluation

8.1 Contracts Compilation

```
scarb --version; scarb build
scarb 2.5.4 (28dee92c8 2024-02-14)
cairo: 2.5.4 (https://crates.io/crates/cairo-lang-compiler/2.5.4)
sierra: 1.4.0

   Compiling lib(pyth) pyth v0.1.0 (.../pyth-crosschain/target_chains/starknet/contracts/Scarb.toml)
   Compiling starknet-contract(pyth) pyth v0.1.0 (.../pyth-crosschain/target_chains/starknet/contracts/Scarb.toml)
warn: libfunc `nullable_forward_snapshot` is not allowed in the libfuncs list `Default libfunc list`
 --> contract: pyth
help: try compiling with the `experimental` list
 --> Scarb.toml
    [[target.starknet-contract]]
    allowed-libfuncs-list.name = "experimental"

    Finished release target(s) in 17 seconds
```

8.2 Tests Output

```
$ snforge test

   Compiling lib(pyth) pyth v0.1.0 (.../pyth-crosschain/target_chains/starknet/contracts/Scarb.toml)
   Compiling starknet-contract(pyth) pyth v0.1.0 (.../pyth-crosschain/target_chains/starknet/contracts/Scarb.toml)
warn: libfunc `nullable_forward_snapshot` is not allowed in the libfuncs list `Default libfunc list`
 --> contract: pyth
help: try compiling with the `experimental` list
 --> Scarb.toml
    [[target.starknet-contract]]
    allowed-libfuncs-list.name = "experimental"

    Finished release target(s) in 14 seconds


Collected 61 test(s) from pyth package
Running 16 test(s) from src/
[PASS] pyth::util::tests::test_u32_as_i32 (gas: ~2)
[PASS] pyth::byte_buffer::tests::byte_array_3_items (gas: ~2)
[PASS] pyth::byte_buffer::tests::byte_array_3_bytes (gas: ~2)
[PASS] pyth::byte_buffer::tests::byte_array_3_zeros (gas: ~2)
[PASS] pyth::byte_buffer::tests::byte_array_last_zero_invalid (gas: ~2)

Success data:
    "assertion failed: `num_last_bytes > 0`."

[PASS] pyth::byte_buffer::tests::byte_array_last_too_large (gas: ~2)

Success data:
    "assertion failed: `num_last_bytes <= 31`."

[PASS] pyth::pyth::pyth::test_debug_price_feed_updated (gas: ~9)
[PASS] pyth::byte_buffer::tests::byte_array_empty_invalid (gas: ~2)

Success data:
    "assertion failed: `num_last_bytes == 0`."

[PASS] pyth::byte_buffer::tests::empty_byte_array (gas: ~1)
[PASS] pyth::byte_buffer::tests::byte_array_two_full (gas: ~2)
[PASS] pyth::pyth::price_update::test_debug_price_info (gas: ~12)
[PASS] pyth::pyth::price_update::test_debug_price_feed_message (gas: ~16)
[PASS] pyth::byte_buffer::tests::byte_array_single_full (gas: ~2)
[PASS] pyth::byte_buffer::tests::byte_array_last_too_many_bytes (gas: ~3)

Success data:
    "ByteBufferImpl::new: last value is too large"

[PASS] pyth::pyth::interface::test_debug_price (gas: ~9)
[PASS] pyth::util::tests::test_u64_as_i64 (gas: ~2)
Running 45 test(s) from tests/
[IGNORE] tests::pyth::test_rejects_set_wormhole_without_deploying
[IGNORE] tests::pyth::test_upgrade_rejects_not_pyth
[IGNORE] tests::pyth::test_upgrade_rejects_invalid_hash
[PASS] tests::pyth::test_update_if_necessary_rejects_no_fresh (gas: ~2366)
[PASS] tests::pyth::test_governance_transfer_works (gas: ~4342)
[PASS] tests::pyth::test_update_if_necessary_works (gas: ~4494)
[PASS] tests::pyth::test_rejects_update_after_data_source_changed (gas: ~4619)
[PASS] tests::pyth::test_governance_set_wormhole_works (gas: ~7216)
[PASS] tests::pyth::test_rejects_price_update_without_setting_wormhole (gas: ~3678)
[PASS] tests::pyth::test_governance_set_data_sources_works (gas: ~5403)
[PASS] tests::pyth::test_rejects_set_wormhole_with_incompatible_guardians (gas: ~3433)
[PASS] tests::pyth::test_get_update_fee_rejects_unsupported_token (gas: ~32322)
[PASS] tests::pyth::test_get_single_update_fee_rejects_unsupported_token (gas: ~32318)
[PASS] tests::pyth::test_governance_set_fee_works (gas: ~5258)
[PASS] tests::wormhole::test_submit_guardian_set_rejects_invalid_emitter (gas: ~1056)
[PASS] tests::pyth::test_getters_work (gas: ~32336)
[PASS] tests::pyth::test_set_fee_rejects_wrong_emitter (gas: ~2364)
[PASS] tests::pyth::test_rejects_if_both_fees_insufficient (gas: ~32985)
[PASS] tests::pyth::test_accepts_secondary_fee (gas: ~40842)
[PASS] tests::pyth::test_governance_set_fee_in_token_works (gas: ~5275)
```

```
[PASS] tests::pyth::test_upgrade_works (gas: ~2970)
[PASS] tests::pyth::test_rejects_old_emitter_after_transfer (gas: ~3757)
[PASS] tests::pyth::test_parse_price_feed_updates_works (gas: ~40837)
[PASS] tests::pyth::test_accepts_secondary_fee_if_first_balance_insufficient (gas: ~41059)
[PASS] tests::pyth::test_upgrade_rejects_wrong_magic (gas: ~2447)
[PASS] tests::pyth::test_accepts_secondary_fee_if_first_allowance_insufficient (gas: ~40987)
[PASS] tests::wormhole::test_submit_guardian_set_rejects_wrong_index_in_payload (gas: ~1062)
[PASS] tests::pyth::update_price_feeds_works (gas: ~40846)
[PASS] tests::wormhole::test_submit_guardian_set_rejects_empty (gas: ~1057)
[PASS] tests::wormhole::test_get_guardian_set_rejects_invalid_index (gas: ~2683)
[PASS] tests::wormhole::test_deploy_rejects_empty (gas: ~295)
[PASS] tests::pyth::test_get_no_older_works (gas: ~40849)
[PASS] tests::wormhole::test_submit_guardian_set_rejects_wrong_index_in_signer (gas: ~2896)
[PASS] tests::wormhole::test_submit_guardian_set_emits_events (gas: ~11961)
[PASS] tests::wormhole::test_parse_and_verify_vm_works (gas: ~38193)
[PASS] tests::wormhole::test_get_guardian_set_works (gas: ~30567)
[PASS] tests::pyth::test_update_if_necessary_rejects_empty (gas: ~2366)
[PASS] tests::pyth::test_parse_price_feed_updates_rejects_bad_price_id (gas: ~33410)
[PASS] tests::pyth::test_parse_price_feed_updates_rejects_out_of_range (gas: ~33410)
[PASS] tests::pyth::test_parse_price_feed_updates_unique_works (gas: ~40836)
[PASS] tests::pyth::test_parse_price_feed_updates_unique_rejects_non_unique (gas: ~33416)
[PASS] tests::wormhole::test_submit_guardian_set_rejects_non_governance (runs: 256, gas: {max: ~28911, min: ~28911,
↪  mean: ~28911.00, std deviation: ~0.00})
[PASS] tests::wormhole::test_submit_guardian_set_rejects_corrupted (runs: 100, gas: {max: ~1250, min: ~514, mean:
↪  ~1203.00, std deviation: ~158.85})

Success data:
    0x496e76616c6964207369676e6174757265 ('Invalid signature')

[PASS] tests::wormhole::test_parse_and_verify_vm_rejects_corrupted_vm (runs: 100, gas: {max: ~38267, min: ~30572, mean:
↪  ~34412.00, std deviation: ~2346.66})

Success data:
    0x4f7074696f6e3a3a756e77726170206661696c65642e ('Option::unwrap failed.')

[PASS] tests::pyth::test_rejects_corrupted_governance_instruction (runs: 100, gas: {max: ~2977, min: ~2353, mean:
↪  ~2850.00, std deviation: ~238.38})

Success data:
    0x4f7074696f6e3a3a756e77726170206661696c65642e ('Option::unwrap failed.')

Tests: 58 passed, 0 failed, 0 skipped, 3 ignored, 0 filtered out
Fuzzer seed: 1712529273147471288
```

9 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.